# CÁC THÀNH PHẦN TRONG NGÔN NGỮ C

## ĐỊNH DANH VÀ TỪ KHÓA

### Định danh

Định danh là tên được đặt cho các thực thể như biến, hàm, cấu trúc, … Định danh phải là duy nhất. Chúng được tạo ra để đặt một tên duy nhất cho một thực thể để xác định nó trong quá trình thực thi chương trình. Định danh phải khác với từ khóa.

|  |
| --- |
| **VD:** Một số định danh |
| int money;  double accountBalance; |

#### Bảng chữ cái

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ kí tự nào đó và các quy tắc trên đó để xây dựng các từ, các câu lệnh và cấu trúc chương trình. Ngôn ngữ lập trình C sử dụng bộ ký tự ASCII (American Standard Code for Informations Interchange). Theo chuẩn này, bộ kí tự gồm có 256 kí tự đó là:

* Các chữ cái: A … Z, a .. z
* Các chữ số: 0 1 2 3 4 5 6 7 8 9
* Các dấu phép toán số học: +,-,\*,/,...
* Các ký tự đặc biệt trong lập trình C

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Các ký tự đặc biệt trong lập trình C** | | | | |
| , | < | > | . | ~ |
| ( | ) | ; | $ | : |
| % | [ | ] | # | ? |
| ' | & | { | } | " |
| ^ | ! | \* | / | | |

Mỗi kí tự có tương ứng 1 số duy nhất gọi là mã, trong đó có 128 kí tự đầu (có mã từ 0 tới 127) là kí tự cố định và 128 kí tự còn lại (có mã từ 128 tới 255) là các kí tự mở rộng, tức là nó có thể thay đổi tuỳ theo ngôn ngữ mỗi quốc gia sử dụng.

#### Khoảng trắng

Khoảng trắng là thuật ngữ được sử dụng trong C để mô tả khoảng trống, tab, ký tự dòng mới và nhận xét. Khoảng trắng phân tách một phần của một câu lệnh với phần khác và cho phép trình biên dịch xác định vị trí của một phần tử trong một câu lệnh, chẳng hạn như int, kết thúc và phần tử tiếp theo bắt đầu.

|  |
| --- |
| **VD:** Trong câu lệnh sau, có ít nhất một ký tự khoảng trắng giữa int và age để trình biên dịch có thể phân biệt chúng. |
| int age; |
| **VD:** Trong câu lệnh sau, không có ký tự khoảng trắng nào giữa fruit và =, hoặc giữa = và apples, mặc dù bạn có thể thêm một số ký tự nếu muốn tăng khả năng đọc. |
| fruit=apples + oranges; |

#### Quy tắc đặt định danh

Định danh rất quan trọng trong quá trình lập trình, nó không những thể hiện rõ ý nghĩa trong chương trình mà còn dùng để xác định các đối tượng khác nhau trong chương trình.

* Chiều dài tối đa của định danh là 32 ký tự.
* Định danh hợp lệ là một chuỗi ký tự liên tục gồm:
* Ký tự chữ, số và dấu gạch dưới.
* Ký tự đầu phải là chữ hoặc dấu gạch dưới.
* Không được đặt trùng với các từ khóa.
* Định danh nên được đặt tên theo quy tắc Hungarian Notation, CameCase.

**Lưu ý:** Trong ngôn ngữ C có phân biệt chữ hoa, chữ thường.

|  |  |
| --- | --- |
| **VD:** Một số định danh đúng trong ngôn ngữ C. | |
| **Mã chương trình** | **Ý nghĩa** |
| int hocSinh; | Biến hocSinh có kiểu dữ liệu int |
| int Tuoi; | Biến Tuoi có kiểu dữ liệu int |

|  |  |  |
| --- | --- | --- |
| **VD:** Tên biến phân biệt chữ hoa và chữ thường. | | |
| **Mã chương trình** | | **Ý nghĩa** |
| number | Number | Biến number khác biến Number |
| case | Case | case là từ khóa, Case vẫn đúng |

***1. Quy tắc Hungarian Notation***

**VD**: Ta có biến Tuoi có kiểu dữ liệu int, khi sử dụng quy tắc Hungarian Notation bạn thêm chữ i (kí tự đầu của kiểu dữ liệu) vào đầu tên biến Tuoi để trong quá trình lập trình hoặc sau này xem lại, sửa chữa… bạn dễ dàng nhận ra biến Tuoi có kiểu int mà không cần phải di chuyển đến phần khai báo mới biết kiểu của biến này.

|  |
| --- |
| **VD:** Khai báo biến theo quy tắc ***Hungarian Notation*** |
| int ihocSinh;  double dlaiSuat; |

***2. Quy tắc CameCase***

Quy tắc CameCase nghĩa là nếu tên biến có nhiều hơn hai từ thì bắt đầu từ thứ hai trở đi của tên biến sẽ lên hoa.

|  |
| --- |
| **VD:** Khai báo biến theo quy tắc ***CameCase.*** |
| int ihocSinhGioi;  double dlaiSuat; |

### Từ khóa

Từ khóa là các từ dành riêng được sử dụng trong lập trình có ý nghĩa đặc biệt đối với trình biên dịch đã được xác định trước. Từ khóa là một phần của cú pháp và chúng không thể được sử dụng làm hằng số hoặc biến hoặc bất kỳ tên định danh nào khác.

**Lưu ý**: Ngôn ngữ C phân biệt chữ hoa chữ thường, nên tất cả các từ khóa phải được viết bằng chữ thường.

|  |  |
| --- | --- |
| **Mã chương trình** | **Ý nghĩa** |
| int money; | int là một từ khóa cho biết biến money là một biến kiểu int. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Danh sách từ khóa** | | | | | | | |
| auto | continue | double | for | int | signed | struct | void |
| break | do | else | if | long | static | switch | while |
| case | default | enum | goto | register | sizeof | typedef | volatile |
| char | const | extern | float | return | short | union | unsigned |

## COMMENTS TRONG C

### Comment trong C là gì

**Comment trong C** hay còn gọi là cách **chú thích trong C** hoặc là **ghi chú trong C** là các dòng code được bỏ qua khi chạy chương trình, nhằm giúp bạn lưu lại các thông tin khi viết chương trình.

### Các cách comment trong C

Có 2 cách comment trong D, đó là:

* Comment trên một dòng trong C. Sử dụng: //
* Comment trên nhiều dòng trong C. Sử dụng: /\* \*/

#### Comment trên một dòng trong C

Chúng ta sử dụng **//** để comment một dòng trong C. Comment trong C sẽ được tính từ vị trí bắt đầu // cho tới hết dòng.

|  |
| --- |
| //Đây là Comment |

Chúng ta có thể bắt đầu comment bằng cách ghi // từ vị trí đầu dòng, hay ở giữa dòng đều được.

|  |
| --- |
| #include <stdio.h>  int main(void){      // Người tạo: Xuân Thức      int num;      num = 10;      num = num + 5;  // cộng thêm 5 đơn vị      printf("num = %d\n",num);  // in ra màn hình  } |

Trong ví dụ trên, các dòng dưới đây chính là comment trong C

|  |
| --- |
| // người tạo: Xuân Thức  // cộng thêm 5 đơn vị  // in ra màn hình |

#### Comment trên nhiều dòng trong C

Để comment trên nhiều dòng trong C, ngoài cách ghi vào đầu từng dòng dấu // như ở cách trên, thì chúng ta có thể dùng một cách đơn giản hơn nữa, đó là viết các dòng chú thích ở giữa cặp dấu /\* và \*/ với cú pháp sau đây:

|  |
| --- |
| /\*      Dòng comment 1      Dòng comment 2      …  \*/ |

Chúng ta sử dụng cặp dấu /\* và \*/ để **comment trên nhiều dòng trong C** như ví dụ sau đây:

|  |
| --- |
| #include <stdio.h>  int main(void){      /\*      Ngày viết : 21-03-2023      Tác giả : Xuân Thức    \*/      int num = 10;      num = num + 5;  /\* cộng thêm 5 đơn vị\*/      printf("num = %d\n",num);// in ra màn hình  } |

**Kết quả:**

|  |
| --- |
| 15 |

### Phím tắt comment trong Công cụ

Tùy thuộc vào phần mềm viết code C mà bạn dùng thì chúng ta sẽ có các cách dùng phím tắt khác nhau như dưới đây:

#### Phím tắt comment trong Dev C++

Sau khi chọn các dòng code, chúng ta có thể sử dụng các tổ hợp phím sau để comment trong Dev C++:

* **Ctrl +.** Chuyển vùng code được chọn thành comment
* **Ctrl +,** Bỏ commet của vùng code được chọn

#### Phím tắt comment trong Notepad ++

Sau khi chọn các dòng code, chúng ta có thể sử dụng các tổ hợp phím sau để comment trong Notepad ++:

* **Ctrl + K**: Chuyển vùng code được chọn thành comment
* **Ctrl + Q**: Bỏ commet của vùng code được chọn

#### Phím tắt comment trong Visual Studio

Sau khi chọn các dòng code, chúng ta có thể sử dụng các tổ hợp phím sau để comment trong Visual Studio:

* **Ctrl + K + C**: Chuyển vùng code được chọn thành comment
* **Ctrl + K + U**: Bỏ commet của vùng code được chọn

#### Phím tắt comment trong Sublime text 3

Sau khi chọn các dòng code, chúng ta có thể sử dụng các tổ hợp phím sau để comment trong Sublime text 3:

* **Ctrl + /**: Chuyển vùng code được chọn thành comment
* **Ctrl + Shift + /**: Bỏ commet của vùng code được chọn

## KIỂU DỮ LIỆU

### Kiểu dữ liệu

Kiểu dữ liệu của một biến còn thể hiện dung lượng không gian mà nó chiếm trong bộ nhớ và cách diễn giải mẫu bit được lưu trữ.

### Phân loại kiểu dữ liệu

Có 3 loại kiểu dữ liệu khác nhau trong ngôn ngữ C.

|  |  |  |
| --- | --- | --- |
| **STT** | **Loại kiểu dữ liệu** | **Mô tả** |
| 1 | Loại Cơ bản | Số nguyên, số thực, ký tự |
| 2 | Loại bool | Đúng hay sai |
| 3 | Loại void | Giá trị trống |

#### Kiểu dữ liệu cơ bản

Ngôn ngữ C có 4 kiểu dữ liệu cơ bản là: char, interger, float, double

|  |  |
| --- | --- |
| **Kiểu dữ liệu cơ bản** | |
| **Tên** | **Kích thước (Byte)** |
| char | 1 |
| int | 4 |
| float | 4 |
| double | 8 |

Kích thước trong bộ nhớ và miền giá trị của các kiểu dữ liệu còn phụ thuộc vào hệ thống và chương trình dịch tương ứng. Giá trị được đưa ra ở đây là trên hệ thống Windows 64 bit và trình dịch GCC MinGW.

##### Kiểu số nguyên

Trong ngôn ngữ C, có 4 bổ ngữ để sửa đổi kiểu dữ liệu. Các bổ ngữ này kết hợp với các kiểu dữ liệu chính để phân loại phạm vi sử dụng Tối ưu bộ nhớ lưu trữ.

|  |  |  |
| --- | --- | --- |
| **Các bổ ngữ** | **Mục đích** | **Mô tả** |
| signed | Thay đổi khoảng giá trị của byte. | Có dấu |
| unsigned | Không dấu |
| long | Thay đổi giá trị byte. | 4 byte |
| short | 2 byte |

**Bảng:** Giá trị kiểu dữ liệu số nguyên.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên** | **Mô tả** | **Kích thước** | **Khoảng giá trị** |
| char | Số nguyên bé  (Không nên dùng để lưu số nguyên) | 1 byte | signed: -  unsigned: 0 |
| short | Số nguyên ngắn | 2 byte | signed: -  unsigned: 0 |
| int | Số nguyên | 4 byte | signed: -  unsigned: 0 |
| long | Số nguyên dài | 4 byte | signed: -  unsigned: 0 |
| long long | Số nguyên cực dài | 8 byte | signed: -  unsigned: 0 |

Ngoài ra, một số kiểu dữ liệu khác như:

|  |  |  |
| --- | --- | --- |
| **Tên** | **Kích thước** | **Khoảng giá trị** |
| uint8\_t | 1 byte | 0 |
| uint16\_t | 2 byte | 0 |
| uint32\_t | 4 byte | 0 |

##### Kiểu số thực

Kiểu số thực được dùng để chứa những số có dấu phẩy động.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên** | **Kích thước** | **Khoảng giá trị** | **Độ chính xác** |
| float | 4 | 1.2E-38 to 3.4E+38 | Độ chính xác 6 con số |
| double | 8 | 2.3E-308 to 1.7E+308 | Độ chính xác 15 con số |
| long double | 16 | 3.4E-4932 to 1.1E+4932 | Độ chính xác 19 con số |

##### Kiểu ký tự

Kiểu ký tự dùng để chứa những ký tự.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên** | **Mô tả** | **Kích thước** | **Khoảng giá trị** |
| char | Kí tự | 1 byte | - |
| unsigned char | 0 |

#### Kiểu bool

Giá trị 1: là đúng ( true )

Giá trị 0: là sai (false)

#### Kiểu void

Kiểu void có nghĩa là “không có giá trị”, nó không được dùng trong khai báo biến thông  
thường mà được sử dụng để chỉ định kiểu của các hàm không có giá trị trả về. Xác định kích thước kiểu dữ liệu

#### Toán tử sizeof là gì?

Toán tử sizeof nhận một tham số là bất kỳ kiểu dữ liệu nào và trả về kích thước của kiểu dữ liệu đó.

**Cú pháp:**

|  |
| --- |
| sizeof (Kiểu dữ liệu) |

#### Kiểm tra kích thước các kiểu dữ liệu

|  |
| --- |
| **VD:** Xác định kích thước của các kiểu dữ liệu cơ bản trong C. |
| #include<stdio.h>  int main() {      printf("Kich thuoc kieu char: %i\n", sizeof(char));      printf("Kich thuoc kieu int: %i\n", sizeof(int));      printf("Kich thuoc kieu float: %i\n", sizeof(float));      printf("Kich thuoc kieu double: %i\n", sizeof(double));      return 0;  } |

**Kết quả:**

|  |
| --- |
| Kich thuoc kieu char: 1  Kich thuoc kieu int: 4  Kich thuoc kieu float: 4  Kich thuoc kieu double: 8 |

**VD:** Xác định kích thước của các kiểu dữ liệu **– Mở rộng thêm**

|  |
| --- |
| //Xác định kích thước của các kiểu dữ liệu  #include <stdio.h>      int main() {      char c;      short s;      int i;      unsigned int ui;      float f;      double d;      long l;      long long ll;      unsigned long long ull;      // Lấy kích thước từng kiểu dữ liệu//      printf("Size cua kieu char: %i\n",sizeof(c));      printf("Size cua kieu short: %i\n",sizeof(s));      printf("Size cua kieu int: %i\n",sizeof (i));      printf("Size cua kieu unsigned int: %i\n",sizeof (ui));      printf("Size cua kieu float: %i\n",sizeof (f));      printf("Size cua kieu double: %i\n",sizeof (d));      printf("Size cua kieu long: %i\n",sizeof (l));      printf("Size cua kieu long long: %i\n",sizeof (ll));      printf("Size cua kieu unsigned long long: %i\n",sizeof (ull));      return 0;  } |

**Kết quả:**

|  |
| --- |
| Size cua kieu char: 1  Size cua kieu short: 2  Size cua kieu int: 4  Size cua kieu unsigned int: 4  Size cua kieu float: 4  Size cua kieu double: 8  Size cua kieu long: 4  Size cua kieu long long: 8  Size cua kieu unsigned long long: 8 |

**VD:** Xác định kích thước max, min của kiểu dữ liệu số nguyên

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  #include <float.h>  int main() {      printf("\n------------Signed -------------\n");      printf("char      - Bit: %-20lli\n", CHAR\_BIT);      printf("char      - max: %-20lli - min: %-20lli\n",CHAR\_MAX, CHAR\_MIN);      printf("short     - max: %-20lli - min: %-20lli\n",SHRT\_MAX, SHRT\_MIN);      printf("int       - max: %-20lli - min: %-20lli\n",INT\_MAX, INT\_MIN);      printf("long      - max: %-20lli - min: %-20lli\n",LONG\_MAX, LONG\_MIN);      printf("long long - max: %-20lli - min: %-20lli\n",LLONG\_MAX, LLONG\_MIN);      printf("\n------------Unsigned -------------\n");      printf("unsigned char      - max: %-20llu\n",UCHAR\_MAX);      printf("unsigned short     - max: %-20llu\n",USHRT\_MAX);      printf("unsigned int       - max: %-20llu\n",UINT\_MAX);      printf("unsigned long      - max: %-20llu\n",ULONG\_MAX);      printf("unsigned long long - max: %-20llu\n",ULLONG\_MAX);      return 0;  } |

**Kết quả:**

|  |
| --- |
| ------------signed -------------  char - Bit: 8  char - max: 127 - min: 4294967168  short - max: 32767 - min: 4294934528  int - max: 2147483647 - min: 2147483648  long - max: 2147483647 - min: 2147483648  long long - max: 9223372036854775807 - min: -9223372036854775808  ------------unsigned -------------  unsigned char - max: 255  unsigned short - max: 65535  unsigned int - max: 4294967295  unsigned long - max: 4294967295  unsigned long long - max: 18446744073709551615 |

Tệp tiêu đề float.h xác định các macro cho phép bạn sử dụng các giá trị này và các chi tiết khác về biểu diễn nhị phân của số thực trong chương trình của bạn. Ví dụ sau sẽ in không gian lưu trữ được thực hiện bởi một kiểu float và các giá trị phạm vi của nó:

**VD:** Xác định kích thước max, min của kiểu dữ liệu số thực

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  #include <float.h>  int main() {      printf("float       - max: %-20g - min: %-20g\n",FLT\_MAX, FLT\_MIN);      printf("double      - max: %-20g - min: %-20g\n",DBL\_MAX, DBL\_MIN);      printf("long double - max: %-20g - min: %-20g\n",LDBL\_MAX, LDBL\_MIN);      return 0;  } |

**Kết quả:**

|  |
| --- |
| float - max: 3.40282e+038 - min: 1.17549e-038  double - max: 1.79769e+308 - min: 2.22507e-308  long double - max: 3.20528e-317 - min: 3.20528e-317 |

## BIẾN

### Biến số

#### Khái niệm

Biến (Variable) đơn giản là một thông tin nhỏ được lưu trữ trong RAM. Chúng ta gọi nó là biến số vì nó có thể thay đổi trong quá trình thực hiện chương trình. Biến được nhận dạng thông qua **tên biến và kiểu dữ liệu.**

#### Quy tắc đặt tên biến

Biến số là một định danh nên sẽ tuân theo quy tắc đặt định danh

### Khai báo biến

**Cú pháp:**

|  |
| --- |
| < Kiểu dữ liệu> <Danh sách tên biến>; |

Trong đó

* Kiểu dữ liệu: Là các kiểu dữ liệu cơ bản, bool, …
* Danh sách tên biến: gồm các tên biến có cùng kiểu dữ liệu, mỗi tên biến cách nhau dấu phẩy.

#### Khởi tạo biến

Khởi tạo biến là kết hợp việc vừa khai báo biến với toán tử gán để biến nhận ngay giá trị cùng lúc với khai báo.

**Ví dụ:** Khai báo trước, gán giá trị sau

|  |
| --- |
| void main() {      int a, b, c;      a = 1;      b = 2;      c = 5;  } |

**Ví dụ:** Vừa khai báo vừa gán giá trị (Khởi tạo biến)

|  |
| --- |
| void main() {      int a = 1, b = 2, c = 5;  } |

### Phạm vi của biến

Khi lập trình, bạn phải nắm rõ phạm vi của biến. Nếu khai báo và sử dụng không đúng, không rõ ràng sẽ dẫn đến sai sót khó kiểm soát được, vì vậy bạn cần phải xác định đúng vị trí, phạm vi sử dụng biến trước khi sử dụng biến.

#### Biến toàn cục

* Biến toàn cục được khai báo ở bên ngoài tất cả các hàm.
* Biến toàn cục có thể được truy xuất và sử dụng ở mọi hàm trong chương trình.
* Biến toàn cục được tồn tại cho tới khi chương trình kết thúc.

**VD:** Biến toàn cục

|  |
| --- |
| #include<stdio.h>      char x = 'B';  // Biến toàn cục      int y = 100;   // Biến toàn cục  void Hien\_Thi() {      printf("%c\r\n",x);      printf("%d\r\n",y);  }  int main() {      Hien\_Thi();      return 0;  } |

**Kết quả:**

|  |
| --- |
| B  100 |

#### Biến cục bộ

Là các biến được khai báo trong một hàm. Biến đó sẽ chỉ có thể tồn tại và sử dụng bên trong hàm. Nó sẽ được giải phóng khi hàm kết thúc thực thi.

**VD:** Biến cục bộ

|  |
| --- |
| #include<stdio.h>  int ham\_cong(int x, int y){      int a; // bien cuc bo      int b; // bien cuc bo      a = x;      b = y;      return a+b;  }  int main(){      int ket\_qua;      ket\_qua = ham\_cong(1,2);      printf("Ket qua 1 + 2 = %d\r\n", ket\_qua);      return 0;  } |

**Kết quả:**

|  |
| --- |
| Ket qua 1 + 2 = 3 |

#### Biến static (Biến tĩnh)

##### Biến static trong khai báo biến cục bộ

Khi 1 biến cục bộ được khai báo với từ khóa static. Biến sẽ chỉ được khởi tạo 1 lần duy nhất và tồn tại suốt thời gian chạy chương trình. Giá trị của nó không bị mất đi ngay cả khi kết thúc hàm. Mỗi lần hàm được gọi, giá trị của biến chính bằng giá trị tại lần gần nhất hàm được gọi.

**VD.**

|  |
| --- |
| #include<stdio.h>  int in\_so\_thu\_tu(void){      static int x = 0;      x = x + 1;      printf("%d\r\n",x);  }  int main() {      in\_so\_thu\_tu ();      in\_so\_thu\_tu ();      in\_so\_thu\_tu ();      in\_so\_thu\_tu ();      in\_so\_thu\_tu ();   return 0;  } |

**Kết quả:**

|  |
| --- |
| **1**  **2**  **3**  **4**  **5** |

##### Biến static trong khai báo biến toàn cục và khai báo hàm

Mỗi project thường sẽ được viết trên nhiều File vì mục đích phân chia module cũng như là để dễ bảo trì. Do có nhiều File nên rất có thể ở các File sẽ có sự trùng lặp trong cách đặt tên biến. Để tránh sự cố sai sót này người ta đưa ra khái niệm biến toàn cục tĩnh và hàm tĩnh.

• Biến toàn cục tĩnh sẽ chỉ có thể được truy cập và sử dụng trong File khai báo nó, các File khác không có cách nào truy cập được.

• Hàm tĩnh sẽ chỉ có thể gọi trong File khai báo nó, các File khác không có cách nào gọi hàm này được.

**VD.**

|  |
| --- |
| //-----------------  //A.c  // biến a này chỉ được sử dụng trong file A.c  static int a;  // hàm hienthi() này chỉ được sử dụng trong file A.c  static void hien\_thi() {};  int c;  //------------------  //B.c  // biến a này chỉ được sử dụng trong file B.c  static int a;  // hàm hienthi() này chỉ được sử dụng trong file B.c  static void hien\_thi() {};  int d; |

## HẰNG SỐ

### ****Hằng trong C****

Là một giá trị hằng số không cho phép thay đổi trong quá trình chạy chương trình. Như vậy, chúng ta dùng hằng khi không muốn giá trị bị thay đổi trong suốt thời gian chương trình chạy.

### ****Cách khai báo hằng****

**Cú pháp:**

|  |
| --- |
| **const <kiểu dữ liệu> <tên hằng> = <giá trị>** |

**Ví dụ:**

|  |
| --- |
| const int **MAX** = 100;  const float **PI** = 3.14; |

**Lưu ý:**

* Việc khai báo hằng luôn luôn phải đi kèm với khởi tạo giá trị.
* Ngoài việc sử dụng hằng, chúng ta cũng có thể làm điều tương tự với define trong C (Chúng ta sẽ có 1 bài riêng tìm hiểu về Define).
* Dù sử dụng define hay hằng thì chúng ta vẫn nên viết hoa các chữ cái để đặt tên hằng số.

## NHẬP XUẤT DỮ LIỆU

### Xuất dữ liệu trong C

Trong lập trình C, **printf ( )** là một trong những chức năng đầu ra chính. Hàm gửi đầu ra được định dạng tới màn hình.

**Cú pháp:**

|  |
| --- |
| printf ("chuỗi định dạng",[ đối mục 1, đối mục 2,…]); |

Khi sử dụng hàm phải khai báo tiền xử lý #include

- **printf**: tên hàm, phải viết bằng chữ thường.

- **đối mục 1,…:** là các mục dữ kiện cần in ra màn hình. Các đối mục này có thể là biến, hằng hoặc biểu thức phải được định trị trước khi in ra.

- **chuỗi định dạng**: được đặt trong cặp nháy kép (" "), gồm 3 loại:

+ Đối với chuỗi kí tự ghi như thế nào in ra giống như vậy.

+ Đối với những kí tự chuyển đổi dạng thức cho phép kết xuất giá trị của các đối mục ra màn hình tạm gọi là **mã định dạng**. Sau đây là các dấu mô tả định dạng:

* %c : Ký tự đơn
* %s : Chuỗi
* %d : Số nguyên thập phân có dấu
* %f : Kiểu float - dạng dấu chấm thập phân (decimal notation)
* %e : Kiểu float - dạng lũy thừa (exponential notation)
* %g : Kiểu float ( %f hay %e , khi ngắn hơn)
* %x : Số nguyên hệ 16 không dấu - sử dụng “ABCDEF” (unsigned hexadecimal integer)
* %u : Số nguyên không dấu (unsigned decimal integer)
* %o : Số nguyên hệ 8 không dấu (unsigned octal integer)
* l : Tiền tố dùng kèm với %d, %u, %x, %o để chỉ số nguyên dài (ví dụ %ld)

**+ Các ký tự điều khiển và ký tự đặc biệt**

* \n : Nhảy xuống dòng kế tiếp canh về cột đầu tiên.
* \t : Canh cột tab ngang.
* \r : Nhảy về đầu hàng, không xuống hàng.
* \a : Tiếng kêu bip.
* \\ : In ra dấu \
* \" : In ra dấu "
* \' : In ra dấu '
* %%: In ra dấu %

**Ví dụ 1:**

|  |
| --- |
| printf ("Bai hoc C dau tien. \n"); |

Trong đó: **Bai hoc C dau tien.** Là chuỗi kí tự

**\n** là kí tự điều khiển

**Kết quả:**

|  |
| --- |
| Bai hoc C dau tien.  \_ |

**Ví dụ 2:**

giả sử biến i có giá trị = 5 xuất giá trị biến i

|  |
| --- |
| printf("So ban vua nhap la: %d . \n", i); |

**Trong đó: So ban vua nhap la:, .** Là chuỗi kí tự

**\n** là kí tự điều khiển

**i** đối mục là biến (kiểu int)

**Kết quả:**

|  |
| --- |
| So ban vua nhap la: 5.  **\_** |

### Nhập dữ liệu trong C

Trong lập trình C, scanf() là một trong những chức năng thường được sử dụng để lấy đầu vào từ người dùng. Hàm scanf() đọc đầu vào được định dạng từ đầu vào tiêu chuẩn, chẳng hạn như bàn phím.

**Cú pháp:**

|  |
| --- |
| scanf ("mã định dạng", [ đối mục 1, đối mục 2,…]); |

Khi sử dụng hàm phải khai báo tiền xử lý #include<stdio.h>

- **scanf:** tên hàm, phải viết bằng chữ thường.

- **Mã định dạng:** Những định dạng dùng trong hàm printf() cũng được sử dụng với cùng cú pháp trong hàm scanf()

**- đối mục 1,…:** là danh sách các đối mục cách nhau bởi dấu phẩy, mỗi đối mục sẽ tiếp nhận giá trị nhập vào.

**Danh sách đối mục** trong scanf () phải theo qui tắc:

Đọc giá trị vào một biến có kiểu dữ liệu cơ sở, sử dụng ký hiệu **&** trước tên biến

Đọc giá trị vào một biến có kiểu dữ liệu dẫn xuất, không sử dụng **&** trước tên biến

### Bề rộng trường

Bộ xác định độ rộng trường là bộ xác định định dạng được sử dụng trong các ngôn ngữ lập trình để in đầu ra có định dạng trên màn hình.  
Trong ngôn ngữ lập trình C, các chỉ định định dạng có các loại sau:  
%i hoặc %d cho int  
%c cho char  
%f cho float  
%s cho chuỗi …  
Độ rộng trường như tên gợi ý là độ rộng hoặc khoảng trống cần cung cấp cho trường đó dòng hiển thị cụ thể và nó được chỉ định bằng cách sử dụng số.

#### Bề rộng trường số nguyên (Integer Field Width)

Sử dụng để định dạng đầu ra của dữ liệu kiểu số nguyên.

Ví dụ: %5d sẽ hiển thị số nguyên với bề rộng 5.

VD

|  |
| --- |
| int num = 42;      printf("num = %5d", num); |

Kết quả

|  |
| --- |
| num = \_ \_ \_42 |

Ta thấy phía trước 42 sẽ có 3 kí tự khoảng trắng vậy tổng cộng chúng ta sẽ có 5 kí tự

#### Bề rộng trường số thực (Floating Point Field Width)

Sử dụng để định dạng đầu ra của dữ liệu kiểu số thực.

%6.3f: xuất ra chữ số thực có bề rộng là 6 (tính cả dấu “.”) trong đó có 3 chữ số sau dấu phẩy. Khi sử dụng chuỗi định dạng này, số thực sẽ được hiển thị với bề rộng 6, trong đó 3 chữ số sau dấu thập phân sẽ được hiển thị. Nếu số thập phân không đủ 3 chữ số, các chữ số 0 sẽ được thêm vào bên phải của số thập phân để đạt được độ dài 3 chữ số thập phân. Tuy nhiên khi ta yêu cầu hiển thị ít hơn chữ số thập phân ban đầu, số đó sẽ được làm tròn lên hoặc giữ nguyên theo quy tắc làm tròn

VD

|  |
| --- |
| float pi = 3.14159;      float a = 3.12;      printf("a =%6.3f\n",a);      printf("pi =%6.2f\n", pi);      printf("%.3f", pi); |

Kết quả:

|  |
| --- |
| a =\_3.120  pi =\_ \_3.14  3.142 |

**Lưu ý:** Số đằng sau dấu chấm trong bề rộng trường số thực phải bé hơn số đứng trước

VD: a = 3.12312 chúng ta không thể in ra bằng lệnh

|  |
| --- |
| int a = 3.12312;      printf("a= %3.5f",a); |

**Kết quả**

|  |
| --- |
| a= 0.00000 |

**Tương tự như trên ta có**

#### Bề rộng trường chuỗi (String Field Width)

Sử dụng để định dạng đầu ra của dữ liệu kiểu chuỗi. Ví dụ: %10s sẽ hiển thị chuỗi với bề rộng 10.

#### Bề rộng trường ký tự (Character Field Width)

Sử dụng để định dạng đầu ra của dữ liệu kiểu ký tự. Ví dụ: %4c sẽ hiển thị ký tự với bề rộng 4.

#### Bề rộng trường điểm động (Dynamic Field Width)

Đây là loại bề rộng trường linh hoạt, cho phép định dạng đầu ra của dữ liệu với bề rộng được xác định bởi biến hoặc tham số truyền vào. Ví dụ: %\*d sẽ hiển thị số nguyên với bề rộng được xác định bởi giá trị của biến được truyền vào.

#### Bổ từ

##### 3.6.1 Bổ từ "-" (left-justified)

Sử dụng để căn lề trái cho dữ liệu được hiển thị. Ví dụ: %-5d sẽ hiển thị số nguyên với bề rộng 5 và căn lề trái.

VD

|  |
| --- |
| int num = 10;      printf("num =%-5d", num); |

Kết quả

|  |
| --- |
| num =10\_ \_ \_ |

##### 3.6.2 Bổ từ "0" (zero padding)

Sử dụng để điền các số 0 vào đầu số nguyên để đạt được bề rộng mong muốn. Ví dụ: %05d sẽ hiển thị số nguyên với bề rộng 5 và điền số 0 vào đầu.

VD

|  |
| --- |
| int num = 8;      printf("num=%05d", num); |

Kết quả

|  |
| --- |
| num=00008 |

## DEFINE

### Sự miêu tả

Trong ngôn ngữ lập trình C, chỉ thị #define cho phép định nghĩa các macro trong mã nguồn của bạn. Các định nghĩa macro này cho phép khai báo các giá trị không đổi để sử dụng trong toàn bộ mã của bạn.

Định nghĩa macro không phải là biến và mã chương trình của bạn không thể thay đổi như biến. Bạn thường sử dụng cú pháp này khi tạo các hằng số đại diện cho các số, chuỗi hoặc biểu thức.

### Cú pháp

Cú pháp tạo **Hằng** bằng #define trong ngôn ngữ C là:

|  |
| --- |
| #define CNAME **Giá trị** |
| #define CNAME (**Biểu thức**) |

**CNAME**

Tên của hằng số. Hầu hết các lập trình viên C định nghĩa tên hằng của họ bằng chữ hoa, nhưng đó không phải là yêu cầu của Ngôn ngữ C.

**Giá trị**

Giá trị của hằng số.

**Biểu thức**

Biểu thức có giá trị được gán cho hằng số. Biểu *thức* phải được đặt trong dấu ngoặc đơn nếu nó chứa các toán tử.

**Ghi chú**

* KHÔNG đặt ký tự dấu chấm phẩy ở cuối câu lệnh #define. Đây là một sai lầm phổ biến.

Ví dụ:

Hãy xem cách sử dụng chỉ thị #define với số, chuỗi và biểu thức.

#### Số

Sau đây là một ví dụ về cách bạn sử dụng chỉ thị #define để xác định một hằng số:

|  |
| --- |
| #define AGE 10 |

Trong ví dụ này, hằng số có tên AGE sẽ chứa giá trị là 10

#### Chuỗi

Bạn có thể sử dụng chỉ thị #define để định nghĩa một hằng chuỗi.

Ví dụ:

|  |
| --- |
| #define NAME "TechOnTheNet.com" |

Trong ví dụ này, hằng số được gọi là NAME sẽ chứa giá trị của "TechOnTheNet.com".

Dưới đây là một chương trình C ví dụ nơi chúng tôi xác định hai hằng số này:

|  |
| --- |
| #include <stdio.h>  #define NAME "TechOnTheNet.com"  #define AGE 10  int main()  {  printf("%s is over %d years old.\n", NAME, AGE);  return 0;  } |

Chương trình C này sẽ in như sau:

|  |
| --- |
| TechOnTheNet.com is over 10 years old. |

#### Biểu thức

Bạn có thể sử dụng lệnh #define để xác định một hằng số bằng một biểu thức.

Ví dụ:

|  |
| --- |
| #define AGE (20 / 2) |

Trong ví dụ này, hằng số có tên AGE cũng sẽ chứa giá trị là 10.

Dưới đây là một chương trình C ví dụ trong đó chúng tôi sử dụng một biểu thức để xác định hằng số:

|  |
| --- |
| #include <stdio.h>  #define AGE (20 / 2)  int main()  {  printf("TechOnTheNet.com is over %d years old.\n", AGE);  return 0;  } |

Chương trình C này cũng sẽ in như sau:

|  |
| --- |
| TechOnTheNet.com is over 10 years old. |

## TOÁN TỬ

Toán tử là một ký hiệu báo cho trình biên dịch thực hiện các hàm toán học hoặc logic cụ thể. Ngôn ngữ C có nhiều toán tử tích hợp sẵn và cung cấp các loại toán tử sau –

* Toán tử số học
* Toán tử tăng giảm
* Toán tử gán
* Toán tử quan hệ
* Toán tử logic
* Toán tử thao tác trên bit
* Toán tử 3 ngôi

### Toán tử số học

Toán tử số học thực hiện các phép toán như cộng, trừ, nhân, chia, v.v. trên các giá trị số (hằng và biến).

|  |  |
| --- | --- |
| **Toán tử** | **Ý nghĩa của toán tử** |
| + | Phép cộng |
| - | Phép trừ |
| \* | Phép nhân |
| / | Phép chia (Lấy phần nguyên) |
| % | Phép chia lấy dư (Chỉ áp dụng cho số nguyên) |

**thứ tự toán tử số học như sau:**

|  |  |  |
| --- | --- | --- |
|  |  |  |

(%,/,\*);(+,-)

**VD1**:

|  |
| --- |
| 1+2=3  3-2=1  1\*3=3  10/3 = 3  10%3 = 1 |

### Toán tử tăng và giảm

Toán tử ++: Tăng giá trị lên 1 đơn vị.

**• x++:** thực hiện lệnh trước rồi mới tăng x lên 1 đơn vị.

**• ++x:** tăng x lên 1 đơn vị rồi mới thực hiện lệnh.

Toán tử --: Giảm giá trị đi 1 đơn vị.

**• x--:** thực hiện lệnh trước rồi mới giảm x đi 1 đơn vị.

**• --x:** giảm x đi 1 đơn vị rồi mới thực hiện lệnh.

**thứ tự tính toán của toán tử tăng giảm:**

(++x.—x);(x++,x--).

**VD**:

|  |
| --- |
| #include <stdio.h>  int main(){      int x = 10;      int y = 10;      printf("Truoc x = %d\n",++x); // cộng trước rồi thực hiện in      printf("Truoc x = %d\n",x);      printf("Truoc y = %d\n",y++); // in trước rồi thực hiện cộng      printf("Truoc y = %d\n",y);      return 0;  } |

**Kết quả**

|  |
| --- |
| Truoc x = 11  Truoc x = 11  Truoc y = 10  Truoc y = 11 |

**VD:**

|  |
| --- |
| #include <stdio.h>  int main(){      int x = 10;      int y = 10;      printf("Truoc x = %d\n",--x); // trừ trước rồi thực hiện in      printf("Truoc x = %d\n",x);      printf("Truoc y = %d\n",y--); // in trước rồi thực hiện trừ      printf("Truoc y = %d\n",y);      return 0;  } |

**Kết quả**

|  |
| --- |
| Truoc x = 9  Truoc x = 9  Truoc y = 10  Truoc y = 9 |

### Toán tử gán

Toán tử gán được sử dụng để gán giá trị cho một biến. Toán tử gán phổ biến nhất là gán (=)

Toán tử gán có thể kết hợp với toán thử số học.

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ví dụ** | **Được hiểu là** |
| = | a = b | a = b |
| += | a += b | a = a + b |
| -= | a -= b | a = a – b |
| \*= | a \*= b | a = a \* b |
| /= | a /= b | a = a / b |
| %= | a %= b | a = a % b |

**thứ tự ưu tiên của toán tử gán:**

(%=,/=,\*=);(+=,-=),(=).

**VD**

|  |
| --- |
| x += 2; tương đương x = x + 2;  x -= 2; tương đương x = x - 2;  x \*= 2; tương đương x = x \* 2;  x /= 2; tương đương x = x / 2;  x %= 2; tương đương x = x % 2; |

### Toán tử quan hệ

Một toán tử quan hệ kiểm tra mối quan hệ giữa hai toán hạng. Nếu quan hệ là đúng, nó trả về 1 (True); nếu quan hệ là sai, nó trả về giá trị 0 (False).

**Lưu ý:** 1 ở đây là True và 0 ở đây là False, chứ k phải là giá trị 1 và giá trị 0

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ý nghĩa** | **Ví dụ** |
| == | so sánh bằng | 5 == 3 (0) |
| > | so sánh lớn hơn | 5 > 3 (1) |
| < | so sánh bé hơn | 5 < 3 (0) |
| != | so sánh khác | 5 != 3 (1) |
| >= | Lớn hơn hoặc bằng | 5 >= 3 (1) |
| <= | Bé hơn hoặc bằng | 5 <= 3 (0) |

**thứ tự ưu tiên của toán quan hệ là cùng cấp thực hiện từ trái qua phải**

**Ví dụ:** Toán tử quan hệ

|  |
| --- |
| #include <stdio.h>  int main(){      int a = 5, b = 5, c = 10;      printf("%d == %d is %d \n", a, b, a == b);      printf("%d == %d is %d \n", a, c, a == c);      printf("%d > %d is %d \n", a, b, a > b);      printf("%d > %d is %d \n", a, c, a > c);      printf("%d < %d is %d \n", a, b, a < b);      printf("%d < %d is %d \n", a, c, a < c);      printf("%d != %d is %d \n", a, b, a != b);      printf("%d != %d is %d \n", a, c, a != c);      printf("%d >= %d is %d \n", a, b, a >= b);      printf("%d >= %d is %d \n", a, c, a >= c);      printf("%d <= %d is %d \n", a, b, a <= b);      printf("%d <= %d is %d \n", a, c, a <= c);      return 0;  } |

**Kết quả**

|  |
| --- |
| 5 == 5 is 1  5 == 10 is 0  5 > 5 is 0  5 > 10 is 0  5 < 5 is 0  5 < 10 is 1  5 != 5 is 0  5 != 10 is 1  5 >= 5 is 1  5 >= 10 is 0  5 <= 5 is 1  5 <= 10 is 1 |

### Toán tử logic

Một biểu thức chứa toán tử logic trả về 0 hoặc 1 tùy thuộc vào việc biểu thức cho kết quả đúng hay sai. Toán tử logic thường được sử dụng trong việc ra quyết định [trong lập trình C.](https://www.programiz.com/c-programming/c-if-else-statement)

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ý nghĩa** | **Ví dụ** |
| && | Logic AND. Chỉ đúng nếu tất cả các toán hạng là đúng | Nếu c = 5 và d = 2 thì biểu thức ((c==5) && (d>5)) bằng 0 |
| || | Logic OR. Chỉ đúng nếu một trong hai toán hạng là đúng | Nếu c = 5 và d = 2 thì biểu thức ((c==5) || (d>5)) bằng 1. |
| ! | logic NOT. Chỉ đúng nếu toán hạng là 0 | Nếu c = 5 thì biểu thức !(c==5)bằng 0 |

**thứ tự ưu tiên của toán tử logic ngang nhau thực hiện từ trái qua phải**

**VD**

|  |
| --- |
| #include <stdio.h>  int main(){      int a = 5, b = 5, c = 10, result;      result = (a == b) && (c > b);      printf("(a == b) && (c > b) is %d \n", result);      result = (a == b) && (c < b);      printf("(a == b) && (c < b) is %d \n", result);      result = (a == b) || (c < b);      printf("(a == b) || (c < b) is %d \n", result);      result = (a != b) || (c < b);      printf("(a != b) || (c < b) is %d \n", result);      result = !(a != b);      printf("!(a != b) is %d \n", result);      result = !(a == b);      printf("!(a == b) is %d \n", result);      return 0;  } |

**Kết quả:**

|  |
| --- |
| (a == b) && (c > b) is 1  (a == b) && (c < b) is 0  (a == b) || (c < b) is 1  (a != b) || (c < b) is 0  !(a != b) is 1  !(a == b) is 0 |

### Toán tử thao tác trên bit

|  |  |
| --- | --- |
| **Phép thao tác trên bit** | **Kí hiệu** |
| Phép AND | & |
| Phép OR | | |
| Phép phủ định NOT | ~ |
| Phép XOR | ^ |
| Phép dịch trái - Shift left | << |
| Phép dịch phải - Shift right | >> |

**thứ tự ưu tiên của toán tử logic :**

(!);(<<,>>);(^);(|,&)

**Phép AND bit**

Ta có bảng logic của phép AND như sau

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A&B** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Phép OR bit**

Ta có bảng logic của phép OR như sau

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A OR B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Phép NOT bit**

Ta có bảng logic của phép NOT như sau

|  |  |
| --- | --- |
| **A** | **NOT A** |
| 0 | 1 |
| 1 | 0 |

**Phép XOR bit**

Ta có bảng logic của phép XOR như sau

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A XOR B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Phép dịch trái bit**

Khi toán tử dịch trái được thực hiện trên một toán hạng, những bit của toán hạng được dịch về bên trái. Các bit bị chuyển sang trái bị mất và 0 thay vào phía bên phải của toán hạng.

Ta cũng có:

A << n = A \* 2n

**Ví dụ:**

(00000001b) << 1 = (00000010b)

(00001111b) << 2 = (00111100b)

(00111100b) << 4 = (11000000b)

**Phép dịch phải bit**

Khi toán tử dịch phải được thực hiện trên một toán hạng, những bit của toán hạng được dịch về bên phải. Các bit bị chuyển sang phải bị mất và 0 thay vào phía bên trái của toán hạng.

Ta cũng có:

A << n = A / 2n

Ví dụ:

(11000000b) >> 1 = (01100000b)

(11000000b) >>2 = (00110000b)

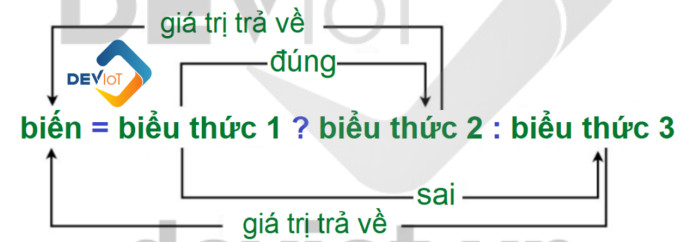
(00111100b) >> 4 = (00000011b)

### Toán tử 3 ngôi

**Cú pháp**

bieu\_thuc\_1 **?** bieu\_thuc\_2 **:** bieu\_thuc\_3;

Nếu như điều kiện ở **bieu\_thuc\_1** đúng thì **bieu\_thuc\_2** trở thành giá trị của toàn bộ biểu thức. Ngược lại nếu như điều kiện ở **bieu\_thuc\_1** sai thì **bieu\_thuc\_3** trở thành giá trị của toàn bộ biểu thức.



|  |
| --- |
| //Ta xét ví dụ sau :  giave = (tuoi < 10) ? 10000 : 50000;  // thứ tự thực hiện  // kiểm tra tuoi<10 đúng không đúng trả về 10000 sai trả về 50000  //biểu thức này tương đương với  if(tuoi < 10){      giave = 10000;  }  else{      giave = 50000;  } |

Như vậy ta thấy việc sử dụng toán tử điều kiện đã giúp thu gọn được đáng kể số lượng dòng code phải không nào.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main(){   // định nghĩa các biến      int n1 = 16, n2 = 8, max;   // Số lớn nhất giữa n1 và n2      max = (n1 > n2) ? n1 : n2;   // In ra số lớn nhất      printf("So lon nhat giua %d va %d la %d.", n1, n2, max);      return 0;  } |

**Kết quả**

|  |
| --- |
| So lon nhat giua 16 va 8 la 16. |

### Ưu tiên toán tử trong C

Thứ tự ưu tiên của toán tử xác định nhóm các thuật ngữ trong một biểu thức và quyết định cách đánh giá một biểu thức. Một số toán tử có quyền ưu tiên cao hơn những toán tử khác; ví dụ, toán tử nhân có độ ưu tiên cao hơn toán tử cộng.

Ví dụ: x = 7 + 3 \* 2; ở đây, x được gán 13, không phải 20 vì toán tử \* có độ ưu tiên cao hơn +, do đó, trước tiên, nó được nhân với 3\*2 rồi cộng thành 7.

Ở đây, các toán tử có mức độ ưu tiên cao nhất sẽ xuất hiện ở đầu bảng, những toán tử có mức độ ưu tiên thấp nhất sẽ xuất hiện ở cuối bảng. Trong một biểu thức, các toán tử có mức độ ưu tiên cao hơn sẽ được đánh giá trước.

Table

Description automatically generated